1.

```
from functools import reduce
def convert_to_decimal(bits):
        exponents = range(len(bits)-1, -1, -1)
        nums = [_________________ for _________ in zip(bits, exponents)]
        return reduce(lambda acc, num: acc + num, nums)
```

```
[ 2 ** val if bit == 1 else 0 for key, val in zip(bits, exponents)]
```

2.

a.

```
[(word, int(num)) for word, num in [line.split(“,”) for line in lines]]
```

b.

```
def unique_characters(sentence):
        return (s for s in sentence)
```

c.

```
def squares_dict(lower_bound, upper_bound)
        return { i : i **2 for i in range(lower_bound, upper_bound + 1)}
```

3.

```
def strip_characters(sentence, chars_to_remove):
        “”.join([char for char in sentence if char not in chars_to_remove])
```

4.

Python does support closures, since having higher level functions is supported in the language through currying. For example:

```
def outer_function(x):
        def inner_function(y):
                return x + y
        return inner_function
```

Here the inner function closes over the variable x, even though it technically looks like it shouldn’t have access to it.

5.
a.)
nth_fibonnaci :: Int -> (Int -> Int) would be the type signature that I think is valid.

c.)
This is not the best annotation because I think it should be more precise for the user to then understand what can be concatenated/added together. There might be some rules about the type conversions that is undergone in python that may yield a different value than expected.

6.
a.)
The c++ type system is not strongly typed. The 123 integer number was converted into binary when the union operation was performed. Thus, the binary approximation was set as the w.f floating point value. C++ in this case allowed for data manipulation and type punning and so we can infer that C++ isn't a strongly typed language.

b.)
There appears to be some form of a type safety checking in place. This prevents access from inactive union fields which results in more predictable behavior. Thus, we can safely conclude that Zig's type system is more strong. In comparison to C++, which has a less strongly typed system, I personally prefer the Zig typing system largely due to the factor of predictability in behavior to create more rigorous code.

7.

a.)

The values of int and Integer are different but not supertypes, subtypes because they are based on the level of precision to represent numbers. Int is for fixed precision and Integer is for an arbitrary precision. I believe that int, integer are subtypes to num, which would be the supertype of the two types. Meanwhile, I also believe that float, fractional is a subtype to num, but they themselves are not supertypes or subtypes of each other because they are also based on the precision level of representation for those numbers.

b.)

They are all different based on the type of value they remove. For example, a div might return a floating point number or an int and it is ultimately represented with an arbitrary value a. Meanwhile, the mod operator also exhibits the same behavior. However, the + operator can perform operations on more general operands. Therefore, the type signature can be more general. The type signature for div and mod must be more strict due to the operations of floor and ceiling as well as representing floating point numbers must restrict what it can accept as input and what it can give to as output.

c.)

I believe that float and int are unrelated but that const int is a subtype of int and const float is a subtype of float. Although int and float both represent numeric types, the way in which they allocate memory to store the value is fundamentally different. Const int essentially is just a specific kind of int and same with const float.